Spinel: An Opportunistic Proxy for Connecting Sensors to the Internet of Things

Benjamin Billet, Valérie Issarny

SPINEL: An Opportunistic Proxy for Connecting Sensors to the Internet of Things

Benjamin Billet
benjamin.billet@inria.fr
MiMove Project Team
Inria

Valérie Issarny
valerie.issarny@inria.fr
MiMove Project Team
Inria

April 17, 2017

Abstract

Nowadays, various static wireless sensor networks (WSN) are deployed in the environment, for many purposes: traffic control, pollution monitoring, etc. The willingness to open these legacy WSNs to the users is emerging, by integrating them to the Internet network as part of the future Internet of Things (IoT), for example in the context of smart cities and open data policies. While legacy sensors can not be directly connected to the Internet in general, emerging standards such as 6LoWPAN are aimed at solving this issue but require to update or replace the existing devices. As a solution to connect legacy sensors to the IoT, we propose to take advantage of the multi-modal connectivity as well as the mobility of smartphones to use phones as *opportunistic proxies*, that is, mobile proxies that opportunistically discover closeby static sensors and act as intermediaries with the IoT, with the additional benefit of bringing fresh information about the environment to the smartphones' owners. However, this requires to monitor the smartphone's mobility and further infer when to discover and register the sensors so as to guarantee the efficiency and reliability of opportunistic proxies. To that end, we introduce and evaluate an approach based on mobility analysis that uses a novel path prediction technique to predict when and where the user is not moving, and thereby serves anticipating the registration of sensors within communication range. We show that this technique enables the deployment of low-cost resource-efficient mobile proxies to connect legacy WSNs with the IoT.
Keywords: Internet of Things, Middleware, Wireless Sensor Network, Opportunistic Proxy, Opportunistic Networking

1 Introduction

Wireless sensor networks (WSN) are able to provide a wide range of useful information about our environment, for various purposes and application domains [1, 2]: public transports, pollution analysis, tracking of goods, home automation, etc. WSNs are basically composed of battery-powered and resource-constrained devices that interact wirelessly in order to achieve highly specific tasks in geographically limited areas, e.g., public transport management in a city. Nowadays, the willingness to open these existing sensor networks to the users is emerging in various contexts, such as smart cities and open data policies, as a step toward the Internet of Things (IoT) vision of a large-scale worldwide network that extends the Internet network to various objects, or Things, able to interact autonomously [2]. From a high level perspective, as presented

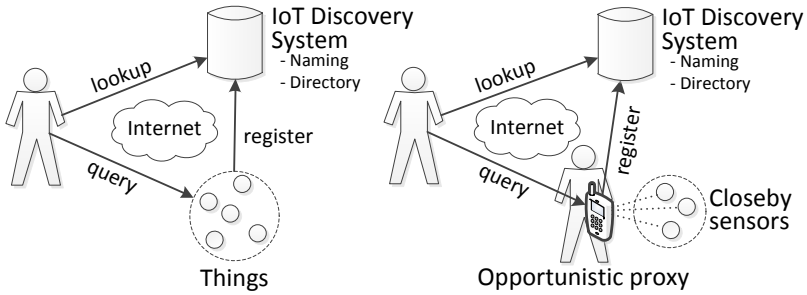

Figure 1: An abstract view of the IoT infrastructure (left) and the role of an opportunistic proxy (right).

in Figure 1, each Thing of the IoT is uniquely identifed by a global *discovery system* composed of two parts: (i) a *naming system* [3] and (ii) a *directory* which can be used to look for devices that match some properties [4]. In addition, each device can be accessed by users, similarly to any node of the Internet.

The new generation of smart Things is able to communicate directly through the Internet, thanks to technological advances of mobile and embedded systems: hardware improvements, new standards, new networking infrastructures, etc. In contrast, existing WSNs rely on resource-constrained motes and various proprietary communication networks and protocols [1]. As a result, new standards are currently emerging to enable small embedded systems to communicate directly with the Internet network. For example, the 6LoWPAN [5] protocol adapts the IPv6 protocol for resource-constrained devices and the CoAP protocol [6] enables the use of RESTful-like services on this class of devices. Unfortunately, deploying these new standards is a difficult task for existing sensor networks, as each device must be updated if possible, or replaced. In addition, although the network layer seems to converge due to the 6LoWPAN standard, physical and link layers, such as 802.11x (e.g., used for Wi-Fi), Bluetooth or 802.15.4 (e.g., used for ZigBee), are not homogeneous. In practice, making a Bluetooth Thing communicate directly with a ZigBee Thing is not possible, and this problem is usually solved by setting up a gateway [1]. However, this solution increases the complexity and the cost of the overall network, as a lot of devices must be deployed to cover the network area. Each sensor's owner has to buy, install and manage these devices, which is not trivial for regular users who just want to contribute to the IoT by opening their sensors.

Consequently, we promote an alternative approach to the networking of Things by leveraging smartphones as *opportunistic proxies* that discover closeby static sensors while moving, register them to the IoT discovery system and forward the users' queries to the related sensors, as shown on the right-hand side of Figure 1. Indeed, smartphones are very good candidates to act as proxies, as they are now equipped with a rich set of advanced communication interfaces like NFC, Bluetooth, Wi-Fi/Wi-Fi Direct and 3G/4G. Some of them are even provided with infrared connectivity (e.g., HTC One) or ZigBee interfaces (e.g., TPH-One). In addition, their mobility and their ever-growing number make them able to cover very large areas. At the same time, smartphone users are looking for more and more fresh and accurate information regarding their environments, e.g., in the context of smart city: by opening their smartphones as proxies, they can improve the quality of the sensed data [7] and benefit from these data in return.

Mobility is a key challenge that must be addressed for enabling opportunistic

proxies, as it has a major impact on the discovery and the registration of closeby sensors. Due to the short connectivity ranges of involved wireless technologies, the set of closeby sensors can vary quickly while the smartphone is moving. As a consequence, the smartphone spends a lot of energy for discovering and registering unnecessary sensors, inducing a strong overload by sending multiple updates to the discovery system, regardless of its level of decentralization. In addition, the same problem affects the queries sent to the network, as smartphones can drift apart from the sensors while a query is processed. All these mobility challenges can be summarized into a single research question: *"How to mitigate the amount of network resources and energy that is consumed during the registration phase by mobile opportunistic proxies for connecting with relevant closeby sensor networks?"*. Such question requires to investigate two subproblems:

1. When the smartphone is not moving at a given location (a pause), the opportunistic proxy needs to (i) infer when it is going to move again, and (ii) check if the user will stay at the same place long enough so as to bear the cost of discovery and registration.

2. In order to reduce the overall energy consumption, the opportunistic proxy should limit the number of interactions with the discovery system so as to minimize the active time of the wireless interfaces.

We then take advantage of human mobility, which is known to have a certain degree of repeatability and predictability [8]. Regarding the first problem, the decision of discovering and registering sensors requires analyzing the common places where the user is not moving and the distribution of pause times. Regarding the second problem, the number of interactions between the proxy and the discovery system can be reduced by leveraging path-based registries [4], i.e., sensor registries able to register a complete path and the sensors available at each step of the path. By registering once a predicted path composed of future pauses' locations and the predicted encountered sensors along this path, there is only one interaction between the proxy and the discovery system over a given period of time.

Following, this paper introduces Spinel, an opportunistic proxy designed to run on smartphones, whose contributions are:

- Spinel reduces the impact of network heterogeneity and integrates existing sensor networks within the IoT without deploying new static gateways or modifying the existing infrastructure. Thanks to the advanced communication capabilities of smartphones (support of several wireless technologies), Spinel dynamically discovers closeby sensors and transparently registers them to the discovery system of the IoT.

- Spinel alleviates the negative impact of mobility, using two mechanisms. First, a process selects the best location sensors for inferring if the user is moving or not, depending on the required accuracy and energy consumption. Knowing the state of the user, Spinel avoids registration of closeby sensors when the user is moving. Second, a path prediction technique, called *Complete Path Prediction* (C2P), analyzes and anticipates users' followed paths. Using the predicted paths and the sensors discovered along these paths in the past, Spinel is able to group registrations, thus reducing the energy/bandwidth consumed for communicating with the discovery system.

This paper is organized in four sections. Section 2 positions our approach with respect to related work. Section 3 presents the Spinel opportunistic proxy and its

interactions with the existing IoT infrastructure, and details the process of mobility analysis and prediction based on our C2P technique. Section 4 evaluates the prototype implementation of SPINEL, by (i) measuring the energy-consumption of the mobility analysis, (ii) evaluating the benefits of the path prediction regarding the number of messages exchanged with the discovery system, and (iii) comparing the results of C2P with another existing technique for trajectory prediction, described in [9]. Finally, Section 5 summarizes our contribution and discusses perspectives for future work.

2 Background

Proxies are very common entities in sensor networks [1, 10, 11], where they are typically used as: (i) centralized data collection and processing points (or sinks) and (ii) gateways between the wireless networks and the Internet. Regarding the latter use, a proxy for WSNs acts as a front-end for the sensor network, receiving messages from the Internet and translating them to the involved sensors and vice versa. Proxies are usually deployed onto dedicated devices (also referred to as base stations) that are more powerful than the regular nodes and are connected to a continuous and reliable power source. As these devices introduce a single point of failure in the network and increase the complexity of the deployment, emerging standards such as 6LoWPAN [5] are aimed at connecting all the small sensors directly using an adapted version of IPv6. However, even in this case, a translation mechanism is required, as 6LoWPAN uses a different address space than IPv6. This problem is solved in practice by integrating the address translation mechanism into the routers that connect 6LoWPAN and IPv6 networks [5]. As 6LoWPAN is not widely used currently and considering that a lot of sensor networks are already deployed in the wild, we argue that smartphones may conveniently be exploited as mobile opportunistic proxies, with the benefit of avoiding the deployment of new devices.

In the context of opportunistic networking, various studies investigate the use of mobility analysis to predict when and where the users are not moving, in order to estimate potential meeting points and the probable duration of the encounter, such as in: opportunistic media sharing [12], content distribution among pedestrians [13] or vehicular networks [14]. In our specific case, mobility prediction is used to (i) reduce the number of messages exchanged between the devices and the registry used for discovery, and (ii) limit the number of data acquisition failures due to mobility of the opportunistic proxies. However, most of the existing work focus on predicting the next future location based on the past mobility of the user, while our approach needs to predict a complete path, i.e., a sequence of next locations, for a long period of time (day or week). As far as we know, long-term path prediction techniques are less studied than the next location prediction techniques. They are typically based on generic statistical models (e.g., Markov models) or human mobility models (e.g., truncated Lévy walk) [4]. Instead of using a statistical model, a small number of approaches, including our work, build a representative set of past paths and try to continuously compare these paths with the current user mobility. Such approaches deal with the problem of path similarity, which consists into finding the distance between two paths. The work presented in [15] introduce a simple and efficient similarity formula for comparing two paths, but it is unclear how representative paths are selected from the set of observed paths and how locations of interest are defined from the GPS traces. The work presented in [9] is based on a more complex DNA sequence alignment techniques for comparing paths. However, this approach is costly in practice and requires to externalize the computation of path similarity values.

With respect to crowdsensing, also known as *participatory sensing* [16] or *oppor-*

tunistic sensing [17] depending on whether the user is involved or not in the sensing process, the opportunistic proxy is intended to sense the environment by acquiring data opportunistically from the closeby wireless sensors. As a benefit, without an actual Internet connection, the user can directly acquire fresh and accurate environmental information. Once obtained, these data are shared with the whole community or a set of selected recipients (e.g., science experiment platforms). For example, in the context of smart cities, users can directly receive pollution-related (e.g., air quality, noise) measurements from closeby sensors.

In scenarios envisioned for the IoT, the number of involved devices can be huge [2] and, given the cost of communication [1], in-network processing should be used instead of pure cloud-based solutions that centrally collect and process sensor measurements and can be affected by unstable connectivity and latency [18, 19]. Interestingly, the proxy avoids the small sensors to be directly requested for in-network processing tasks, by providing an intermediate computation layer between the sensors and the incoming queries. Consequently, our work is also related to code offloading, where devices are able to delegate tasks in order to either increase their battery lifetime or perform complex tasks. For example, *cloudlets* [20, 18, 21] are trusted resource-rich computers where mobile devices can quickly deploy virtual machine and offload expensive computation. *Cyber-foraging* [22] takes advantages of close static and continuously-powered computers, called *surrogates*, to perform resource-consuming tasks. An opportunistic proxy goes a step further, as the surrogate becomes inherently mobile.

3 The Spinel Opportunistic Proxy

By being an interface between existing static WSNs and the IoT, the Spinel opportunistic proxy is assumed to be used in conjunction with existing components of the IoT infrastructure for discovering and querying devices. Consequently, Spinel relies on these components and focuses on its main functions: discovering nearby sensors while moving and opening them to the IoT in an efficient way.

3.1 Interactions with the IoT infrastructure

In a nutshell, the IoT is composed of devices that can communicate through the Internet and interact with each other and their environment (sense and act). Figure 1 shows a high-level description of the IoT infrastructure, where users can look for a set of Things that match their interests and then query them for acquiring data about the environment or trigger an action. To this end, the IoT leverages: (i) a *discovery system* to name, register and look for Things, and (ii) a *querying system* to express, disseminate and process queries over Things.

The discovery system makes it possible to: (i) register sensors and their characteristics into a global *registry*, and (ii) use this registry to find a set, or a representative subset, of sensors that share some specified characteristics (*lookup*). The IoT literature presents various approaches to enable these functionalities, using naming resolution services that associate the name of the Things to a set of characteristics, such as emerging standards like the EPCglobal Object Name Service [23] or uCode [3]. Similarly, lookup services give large-scale solutions for finding a set of sensors that match some given properties, for example the MobIoT [4] middleware or the EPCglobal Discovery Service [23].

Regarding the querying aspects, the supporting system enables to: (i) let the users express queries and inject them into the IoT network, and (ii) process the queries using the data collected from sensors and send back the results to the users. Such

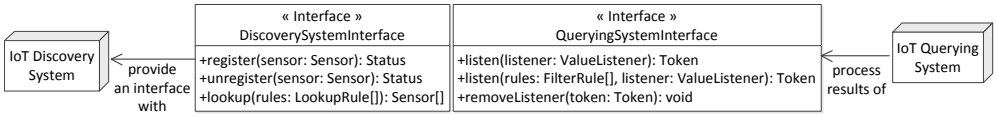

Figure 2: Spinel interfaces for discovery and querying systems.

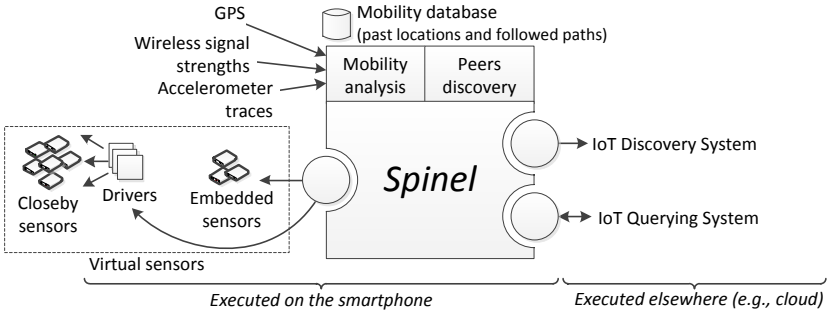

Figure 3: Spinel architecture.

systems already exist in the WSN and IoT literature in the form of middleware for data collection, e.g., TinyDB [24] or Dioptase [25]. These middlewares run on the devices and process the (sub)tasks disseminated in the network.

We define the general roles of an opportunistic proxy as: (i) discover the static sensors that are physically close and make them known to a dedicated registry, and (ii) acquiring data from the closeby sensors in order to answer queries issued by IoT users and autonomous devices. For this purpose, Spinel leverages the work done in the IoT literature, by working in conjunction with the existing parts of the IoT infrastructure mentioned above. Spinel indeed focuses on the opportunistic and mobile aspects, and provides high-level interfaces that enable a developer to provide easily the glue code needed for integrating Spinel with the existing IoT components. As shown in Figure 2, registration and unregistration take a sensor description (address, data type, accuracy, and other metadata) and send it to the discovery system. Lookup takes a set of characteristics and uses the discovery system to find the devices that match them. Finally, the querying system can define a listener for receiving the values collected from sensors by Spinel.

3.2 Architecture Design

The overall architecture of Spinel and its interactions with the IoT systems are shown in Figure 3. To manage the specifics of different classes of wireless sensors and platforms, some low-level functionalities are separated into *drivers*. These drivers are loaded when Spinel starts and are used to access the internal sensors of the smartphone and to list the available communication interfaces.

Both internal and external sensors are abstracted as *virtual sensors*. As a benefit, this avoids the need to modify the existing discovery systems to support proxies: Spinel transparently registers a set of virtual sensors attached to the smartphone and updates this set while new external devices are discovered, as illustrated in Figure 4. In addition, the number of devices to register is reduced as only the smartphone is registered instead of the entire set of closeby sensors: when the registry has to perform a lookup for a given location, the search space is reduced as the number of devices in the area is limited to the number of smartphones only. Finally, the virtual

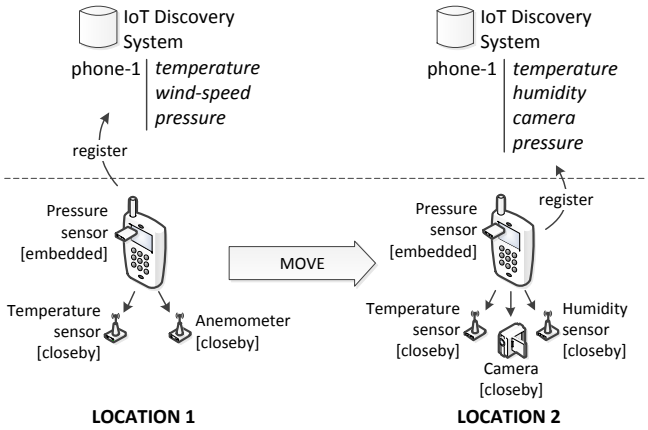

Figure 4: Sensor registration while moving.

sensor abstraction can be used to create software sensors based on various continuous data sources: (i) the data collected by the operating system (CPU and memory consumption, battery level, network throughput, etc.), (ii) user interfaces, enabling the opportunistic proxy to manage human feedback as sensors, and (iii) the APIs provided by smartphones' operating systems for managing external wireless sensors, such as the Android API for acquiring and decoding measurements from e-health Bluetooth sensors. Additional drivers may be added in the future, for example to leverage emerging sensor abstraction layers such as the Google Physical Web (`google.github.io/physical-web`).

In order to reduce the negative effects of mobility, SPINEL leverages a *mobility analysis mechanism* that infers (i) if the user is moving, based on the mobility sensors embedded in the smartphone, and (ii) when the user is going to move after being detected as static, given its past mobility patterns. Unfortunately, acquiring the GPS position of the user can consume a lot of energy in practice [26]. Given that users open their smartphones as opportunistic proxies, the proxy service must be as unobtrusive as possible while consuming a very small part of the smartphone resources. In order to reduce the energy consumption, SPINEL analyzes when and how the smartphones' mobility sensors can be used, depending on their respective energy costs and the required accuracy. In addition, SPINEL continuously builds a *mobility database* that stores past locations and paths followed by the user over time. Depending on these data and the results of the mobility analysis, SPINEL triggers the discovery of the closeby sensors.

The *peers discovery module* encapsulates the APIs provided by the smartphones' operating systems for closeby device discovery. When triggered (i.e., the mobility analysis detects that the user is static enough), the module starts the discovery processes for each available communication interface and, at the end of each process, notifies the set of discovered sensors to SPINEL. These new sensors are registered to the IoT discovery system and their measurements are transmitted to the IoT querying system if required, using the interfaces shown previously in Figure 2.

3.3 Mobility Analysis

Theoretically, the maximum communication range for technologies like Bluetooth and ZigBee does not exceed 100 meters (50 meters for Bluetooth Low Energy). Combined

with the mobility of the smartphones, undesirable behaviors emerge as new sensors are continuously discovered by the proxy while the user is walking, driving, or using public transports. Each lost or newly discovered sensor must be notified to the registry, leading to a potentially huge number of messages exchanged between the proxy and the infrastructure in which the registry is hosted.

SPINEL analyses the smartphone mobility to infer where and when to discover closeby sensors and register them to the discovery system. The mobility analysis process comprises three steps:

- Check if the smartphone is moving, using the embedded mobility sensors by their increasing order of energy-consumption, as shown in Figure 5.
- Acquire the most accurate location of the smartphone (less than five meters, typically provided by the GPS sensor), if not moving.
- Infer how long the smartphone will stay at this location, given past pause times.

In practice, the smartphones' accelerometers consume less energy than the GPS sensors [26]. Consequently, we use the accelerations samples to infer the current activity of the user (walking, driving, not moving, etc.) using existing algorithms for human activity recognition [27] and transportation detection [28]. However, it has to be noted that typical smartphones embed low-cost sensors (accelerometers, gyroscopes, compass, etc.) that may be poorly calibrated or not calibrated at all by the user. In addition, the quality of the measurements produced by such sensors can be affected by several factors, including user sway, external magnetic fields or temperature [29]. As a consequence, the inferred mobility may be strongly inaccurate, meaning that there are two possible outcomes after the recognition phase:

- The algorithm finds the actual mobility state of the user.
- The algorithm produces a false negative, i.e., the user is wrongly detected as static.

Contrary to a false positive, i.e., the user is detected as moving while he is actually not moving, a false negative will trigger the discovery and registration processes and enable the querying system to send data acquisition requests to the smartphone. These operations will thus fail with a high probability, as the discovered sensors will be quickly out of range while the user is moving: for example, if the user is walking, 100 meters will be traveled in approximately two minutes, knowing that the average walking speed of human beings is 5km/h. False negatives can waste a lot of energy, given that a full sensor discovery (packet broadcasting on several network interfaces), a registration (round trip to the registry servers) and, potentially, some data access requests are performed vainly by the querying system. To avoid false negatives, additional location sensors can be used, such as the network location which is based on the signal strength of GSM celltowers or Wi-Fi hotspots. In the worst case where network location does not give enough precision to detect if the user is moving or not, the GPS can be used to acquire some locations and infer the actual speed of the user.

The process for detecting whether the user is moving is presented in Figure 5. Every period of time T_1 (15 minutes, by default), the process is triggered and performs many mobility detection steps, by the increasing order of energy-consumption and accuracy. When a step is not able to infer the user status with the required accuracy, the next step is performed until a reliable result is obtained. If none of the steps produces reliable information, the user is assumed in a moving state.

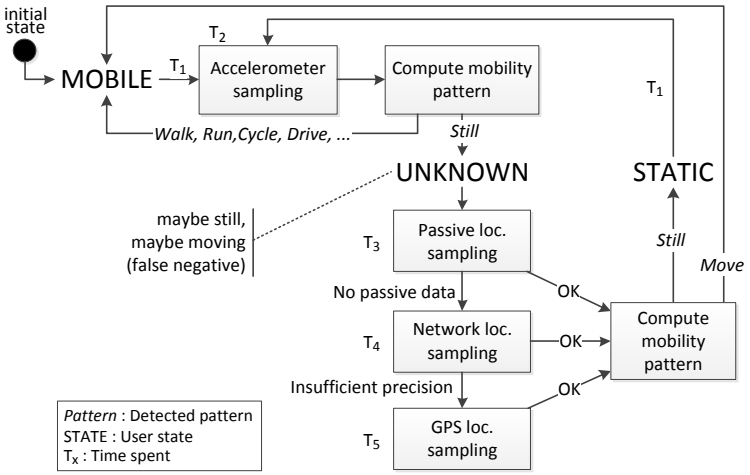

Figure 5: Mobility analysis process.

The first step acquires as much as possible[1] acceleration samples during T_2 minutes and runs the algorithms for human activity recognition and transportation detection. This T_2 time is configurable and must be chosen considering a tradeoff between accuracy and energy consumption. If this step detects that the user is not moving, a false negative can still occur, as activity recognition algorithms are not perfectly reliable [29]. In order to validate the result of the recognition, three other steps can be performed, using the three location sensing modes usually available on smartphones: (i) *passive location* (reuse the location acquired by other applications), (ii) *network location* (based on signal strength wireless networks), and (iii) *GPS location* as a last resort. Similarly to the accelerometer step, many samples are collected for fixed periods of time, respectively denoted T_3, T_4 and T_5. Given that the process is triggered every period of T_1 minutes, the time between two mobility checks is $T_1 + T_2$ in the best case, and $T_1 + T_2 + T_3 + T_4 + T_5$ in the worst one.

When the user is detected as static, Spinel acquires the precise position of the user using the GPS location sensor, if needed. Given this location, two situations are distinguished:

- We do not have information regarding this location in the mobility database of Spinel, and we thus need to take a decision using one of the following strategies: (i) the mobility process detects that the user is static x consecutive times, (ii) the nature of the location or the time of the day is statistically related to a particular pause time, or (iii) a statistical predictor can be used, Spinel providing implementations of predictors commonly found in the literature [30].

- The mobility database contains a location X that is geographically close to the current location of the user, according to the typical range of the wireless sensors (set to 50 meters by default). The existence of X means that the user visited this location in the past and that Spinel measured the pause times. Precisely, Spinel classifies the past pause times at a location in several time classes, i.e., repeatable periods of time (hour of the day, by default). Given the current arrival time of the user at a location, the corresponding time class gives a distribution of pause times in the same time window. If the average of this

[1]The actual number of samples per second is usually managed by the smartphone's operating system.

distribution is greater than a threshold, the discovery and registration processes are triggered.

Using these informed and non-informed decision techniques, communication between the proxy and the registry occurs only when the user is not moving, reducing the number of useless and unneeded registrations.

3.4 Path Prediction

Even though the mobility analysis process detects when it is relevant to discover new sensors and send updates to the discovery system, the number of interactions between the proxy and the registry can be high when the number of users increases: (i) each user at the same location discovers and registers the same sensors (high redundancy), and (ii) more people imply more sensors discovered and more registration messages.

The former problem is easy to solve using a registry able to filter devices that provide similar sensing capabilities, e.g., MobIoT [4]. Specifically, MobIoT estimates when and where two devices with similar sensing capabilities will meet. As a consequence, if more than one smartphone detect the same set of sensors at the same place at the same time, only a subset of the smartphones will register, according to the coverage policies provided to MobIoT by the developers. In practice, MobIoT provides built-in coverage policies that take into account the nature of the sensed physical phenomenon in order to select the most relevant set of smartphones.

The latter problem requires to find a way to reduce the number of updates sent by the proxy to the registry while users are moving from a location to another. We take advantage of both the repeatability of human mobility, where people spend most of their time in a limited number of places [31], and the path-based registries able to register a complete path in a single operation. Concretely, we can predict the next locations of the users for a given period of time, using their past locations. Further, based on the sensors usually discovered at these locations, SPINEL can send a complete predicted path to the registry in a single message. This path will be used by the registry as a scenario played over time. Consequently, after this first registration, the proxy will never communicate with the registry anymore, except for sending correction updates if the prediction is detected as incorrect.

Our path prediction technique is called *Complete Path Prediction* (C2P) as it tries to find a past path that matches the most recent moves of the user, in order to predict the end destination and the intermediary steps for a fixed period of time. Precisely, given $\mathbb{P}$, a set of paths describing the past mobility of a user, and P, a path composed of the last n pauses of the user, C2P looks for a path $Q \in \mathbb{P}$ such that Q is similar to P.

3.4.1 Estimation of Path Similarity

Formally, our approach defines a path and its characteristics as follows:

Définition 1 Path A *path* $P = (p_1, \cdots, p_n)$ is a sequence of n pauses $p = (L, a, t, d, S)$ where L is a location (latitude, longitude), a is the azimuth, t is the arrival time at this location, d is the pause time, and S is the set of sensors discovered at this location.

Similarity between two paths P and Q can be computed by summing the similarity of each pair of pauses (p_i, q_j), expressed as a function that aggregates the individual differences between locations, azimuths, arrival times and pause times.

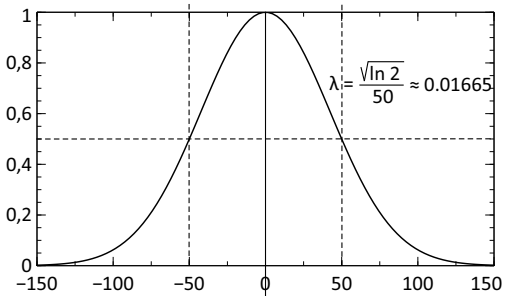

Figure 6: f_d adjusted for producing a similarity of 0.5 for a distance of 50 meters.

Require: P, Q
  $X \leftarrow |P| \times |Q|$ null matrix
  for all $(p_i, q_j) \in P \times Q$ **do**
    $X[i,j] \leftarrow |t_i - t_j|$
  end for
  for all $p_i \in P$ **do**
    $pair \leftarrow \varnothing$, $closeness \leftarrow +\infty$
    for all $q_j \in Q$ **do**
      if $X[i,j] < closeness$ **then**
        $closeness \leftarrow X[i,j]$, $pair \leftarrow (p_i, q_j)$
      end if
    end for
    $pair$ must be evaluated
  end for

Algo. 1: Finding pairs (p_i, p_q) to evaluate.

Définition 2 Spatial dissimilarity *Spatial dissimilarity* of two pauses p and q is given by the geographical distance[2] (in meters) between the locations L_p and L_q, denoted as $\delta(L_p, L_q)$.

Définition 3 Time dissimilarity *Time dissimilarity* of two pauses p and q is given by the differences between arrival times and pause times of p and q, denoted as $\delta(t_p, t_q) = |t_p - t_q|$ and $\delta(d_p, d_q) = |d_p - d_q|$.

Définition 4 Azimuth dissimilarity *Azimuth dissimilarity* of two pauses p and q is given by the difference between azimuths of p and q, denoted as $\delta(a_p, a_q) = |a_p - a_q|$.

Concretely, when these dissimilarity values are high, p and q tend to be dissimilar in terms of space and time. We then define the similarity of two pauses as follows:

Définition 5 Pause similarity *Similarity* of two pauses p and q is given by a value between 0 (p and q are *perfectly dissimilar*) and 1 (p and q are *identical*):

$$f_d(\delta(L_p, L_q)) \cdot f_t(\delta(t_p, t_q)) \cdot f_t(\delta(d_p, d_q)) \cdot \left[\cos \frac{\delta(a_p, a_q)}{2}\right]$$

The purpose of the functions f_d and f_t is to represent the tendency to discover partially different sets of sensors at two locations when their spatial and temporal dissimilarity are low but not null. f_d and f_t are gaussian functions $e^{-(\lambda x)^2}$ that produce a result between 0 and 1, as shown in Figure 6. In practice, any term of the pause similarity formula can be removed/weighted in order to take into account only distance, arrival times, pause times or directions, with customizable f_d and f_t. For example, one of our hypothesis states that if the difference between the arrival times of two pauses is high, these two pauses are probably part of different paths. If this hypothesis is unsuitable in a given scenario, the time dissimilarity can be ignored.

Using the previously stated pause similarity formula, we define the similarity of two paths P and Q as the aggregation of the similarity values of pauses composing P and Q. Given $|P|$ and $|Q|$ the lengths of the paths, two cases must be considered:

- $|P| = |Q|$, in which case the similarity of P and Q is computed by aggregating the individual similarities of pauses (p_i, q_i) $\forall i \in [\![1, |P|]\!]$.
- $|P| \neq |Q|$, in which case we must select which pairs (p_i, q_j) are going to be evaluated.

[2]In practice, the Android OS computes geographical distances using the Vincenty's formulæ [32].

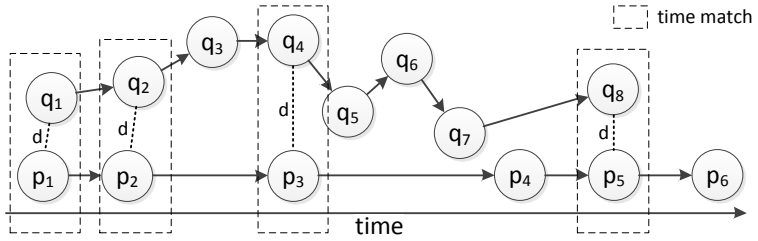

Figure 7: Temporal matching for path similarity (example).

When $|P| \neq |Q|$, we select the pairs of timely close pauses for computing the similarity, as described in Algorithm 1. In the example shown in Figure 7, the time length T_p of P is used as reference time for the comparison, i.e., every node of Q with an arrival time greater than T_p are ignored: the similarity is computed only for the pairs $(p_1, q_1), (p_2, q_2), (p_3, q_4)$ and (p_5, q_8) that form the set of matched pauses M. All the other non-involved pauses are considered having a constant similarity, typically fixed to 0 to represent their dissimilarity with any other pause.

After the execution of the algorithm, all the $k = |P| + |Q| - 2|M|$ similarity values $S = (s_1, ..., s_k)$ are aggregated for computing the similarity value of P and Q. Typically, two aggregation strategies can be used, specifying how the differences between P and Q impact the final result:

Définition 6 Soft aggregation *Soft aggregation* $\tilde{\tilde{S}}(P, Q)$ defines the similarity of two paths P and Q as the average of S values: $\tilde{\tilde{S}}(P, Q) = \sum_{i=1}^{k} \frac{s_i}{k}$

Définition 7 Strict aggregation *Strict aggregation* $\bar{\bar{S}}(P, Q)$ defines the similarity of two paths P and Q as the product of S values: $\bar{\bar{S}}(P, Q) = \prod_{i=1}^{k} s_i$

The soft aggregation reduces the impact of isolated dissimilar pauses, by compensating low similarity values with high similarity values. On the contrary, the strict aggregation considers two paths perfectly dissimilar if two pauses are perfectly dissimilar, which is always the case when $|P| \neq |Q|$. Typically, SPINEL uses strict aggregation to check whether two paths must be merged in the mobility database and soft aggregation for finding a past path that matches the current path followed by the user.

3.4.2 Mobility Database Construction and Lookup

In practice, SPINEL builds the paths progressively using the results of the mobility analysis process that detects when and where the user is static for a significant time, as described in Section 3.3. SPINEL collects and stores paths with different time lengths (day, week, month, etc.). The set $\mathbb{P}$ of paths stored in the mobility database is continuously clustered according to various time resolutions:

Définition 8 Time resolution *Time resolution* of a path P indicates the time length of P and the repeatability of this time length: the day of week (from monday to sunday), the week of the year (from 1 to 53), the day of the month (from 1 to 31), the month of the year (from january to december), etc.

The time resolutions are used to look for repeatable patterns in the user's past mobility, e.g., every monday, every week or every third day of the month. At the end of a time period x (e.g., at the end of the day), each path P of length x are added to the mobility database, with two possible outcomes:

- If there is a path $Q \in \mathbb{P}$ for which $\overline{\overline{S}}(P, Q)$ is greater than a threshold α_1, P and Q are merged, i.e., each pause with close arrival times are merged (average of location, arrival time, azimuth and duration) while temporally isolated pauses are either added or removed of the merged path, depending on SPINEL configuration.

- If there is no path $Q \in \mathbb{P}$ for which $\widetilde{\widetilde{S}}(P, Q)$ is greater than a threshold α_2, P is directly added to $\mathbb{P}$.

Adding a new path P in the relevant cluster is a $O(n)$ operation, where n is the size of the cluster. This is almost negligible, given that this operation is performed only at the end of a time period, the smallest time period being one-day long. Nevertheless, in order to ensure that the mobility database does not grow too big, SPINEL limits the size of each cluster by applying an eviction policy for the stored paths that are either too old (LRU strategy) or not followed frequently enough by the user (LFU strategy).

While the user is moving, SPINEL records the path composed of the last pauses of the user. When the user is detected static at a new location, SPINEL looks for the paths of $\mathbb{P}$ that match the current path followed by the user, for each time period considered (day, week, month, etc.). If there is one or more candidate paths with a soft similarity score greater than a threshold β, the path with the highest score and time length is selected and sent to the registry. The purpose of such a threshold is to reduce the number of wrong predictions sent to the registry. However, once the predicted path is registered, SPINEL still monitors the next pauses of the user and checks if the prediction remains relevant over time. If at some point the prediction appears to be wrong, i.e., if the set of discovered sensors and the predicted set of sensors are different, SPINEL sends a correction update to the registry for this specific location only.

The worst case occurs when the path P predicted at time $t-1$ is completely wrong, because a path P' matches better the last pauses of the user at time t. In this case, SPINEL sends the new predicted path P' to the registry if and only if the strict similarity between P and P' is less than a threshold γ. This last check avoids a costly correction when only one or two pauses are different between P and P'.

3.5 Discussion on security, privacy and incentives

Privacy and security mechanisms are beyond of the scope of this paper, but we have to consider some questions raised by our approach. Privacy concerns arise as the locations of smartphones' owners are either known by the registry system or can be inferred from the sets of closeby sensors that are registered. In this case, we consider that the registry is trusted (similarly to a DNS server) and authenticated, providing strong and reliable encryption for communicating with the smartphones such as no information can be eavesdropped. Nevertheless, additional anonymization techniques can be used, such as onion routing (`www.torproject.org`) or policy-based techniques [33]. In contrast, using an opportunistic proxy can be beneficial for privacy and security from the standpoint of WSNs. The proxy indeed hides the actual sensors by acting as a layer between them and the Internet, and can provide additional mechanisms (encryption, pre-aggretation, etc.) that would not be supported by the sensors, given their limited hardware resources.

The problem of motivating users to open their devices as opportunistic proxies is an issue that must be discussed as well, given that it has an impact on the sensors

coverage, availabity and continuity of access. While it is not possible to give guarantees, as user mobility cannot be forced, various types of incentives were studied in the context of smart city and participatory sensing [7]: financial (e.g., discount coupons), ego-centric (e.g., gamification where the user is rewarded with points, badges, titles, etc.), altruistic (e.g., based on friendliness or kindness) or democratic (e.g., willingness of being better citizens). First, as a direct reward, collecting measurements from closeby sensors directly improves the quality of the information given to the users, especially for phenomenon that occurs around them (i.e., direct reward for the user). Second, as an altruistic or democratic incentive, sharing these information can help to achieve larger goals as well, such as improving the quality of life in a smart city (e.g., pollution analysis) or providing more data for scientific experiments [34]. Third, gamification could be investigated, given that contribution of a user is easy to evaluate: the number of sensors, the amount of data transferred, the number of failures (i.e., the user left the sensor's range while measuring), etc. In addition, this contribution can be evaluated with respect to space (location) and time (recent activity), in order to reward specifically the contributors of a given city or district.

4 Assessment

We have developed a prototype of Spinel which implements the architecture and the algorithms described in Section 3 for the Android platform. As stated before, Spinel works in conjunction with other components of the IoT infrastructure: our prototype provides a connector for the MobIoT [4] path-aware sensor registry and a connector for the Dioptase [25] data streaming and continuous processing middleware, by implementing the interfaces presented in Figure 2.

In practice, the proxy is composed of a background daemon, which monitors the mobility using the mobility analysis process presented in Section 3.3. When the user is detected as static, the precise location is acquired and the discovery of local closeby sensors is triggered. The discovery is performed by using the standard Android connectivity APIs: (i) Bluetooth Service Discovery Protocol (SDP), that returns the set of available Bluetooth devices (addresses, types and UUIDs of available features), (ii) DNS-SD for Wi-Fi/Wi-Fi Direct, that returns a set of WiFi devices (addresses, ports and types of available service), (iii) continuous NFC discovery that returns the set of closeby tags (tags' identifiers and supported technologies).

The protocol-specific results of these discovery processes are abstracted as virtual sensors, as well as the embedded sensors of the smartphone. As described in Section 3.4, the proxy acquires the precise location of the smartphone when the user is not moving and then updates its mobility database. While new locations are reached by the user, the proxy checks the currently running prediction and corrects it – if needed – by sending to the registry the set of old sensors to unregister and the set of new sensors to register. However, if a new path is predicted, the proxy sends the entire new prediction to the registry. If no prediction is active, i.e., there is not enough information to predict a path with a satisfying accuracy, Spinel still sends to the registry the set of new sensors found by the discovery process.

From this implementation, two aspects of our opportunistic proxy are evaluated: (i) the impact of the mobility analysis process on the energy consumption compared to GPS- and accelerometer-only approaches, and (ii) the amount of messages exchanged between Spinel proxies and registries, which must be low in order to reduce both the network load and the active time of the smartphones' wireless interfaces, thereby also limiting the energy consumption.

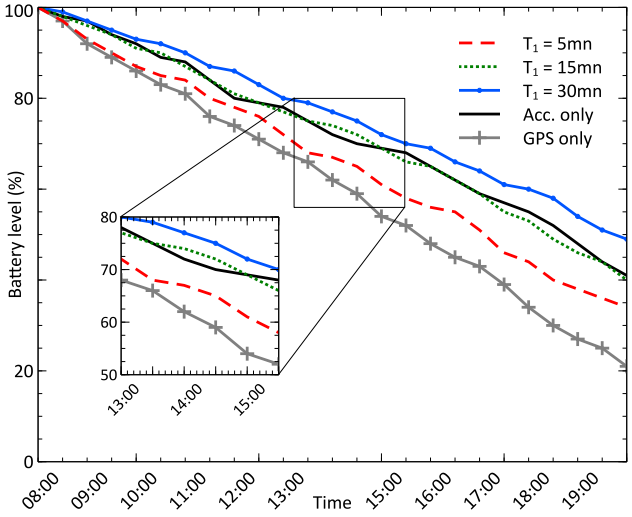

Figure 8: Average battery depletion over a day for different T_1.

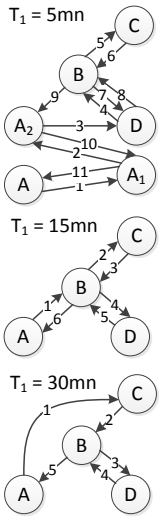

Figure 9: One-day path for different T_1.

Energy Consumption of the Mobility Analysis Process: Except communication with the repository, the most energy-consuming components of SPINEL is the mobility analysis process presented in Section 3.3, which acquires data from the smartphone's location sensors and performs the path prediction. This process infers if the user is moving, using location sensors by order of energy consumption, and we measured how this process affects the battery level compared to accelerometer- and GPS-only mobility detection mechanisms. To this end, we deployed our prototype on a Galaxy Nexus smartphone carried by one user, and we monitored the daily battery depletion. The first day, only the accelerometer was monitored to infer if the user was moving or not. The second day, only the GPS was monitored. The following days, given that the actual consumption of the process depends on the value of T_1, we runned our process for different T_1 (5, 15 and 30 minutes). Given that this experiment targets the mobility analysis process, no energy was spent to communicate with the registry.

The results of the above approaches to the mobility analysis (accelerometer-only, GPS-only, SPINEL with various T_1) are shown in Figure 8, which presents the battery depletion over one day. We can see that the GPS-only based approach is the most battery-consuming while $T_1 = 15$ minutes consumes nearly the same amount of energy than the accelerometer-only technique. Our process has however the advantage of avoiding false negatives, by using the GPS when the mobility can not be inferred from the accelerometer. Given that accelerometer-based detection can produce a significant amount of false negatives, as shown in the literature [29], our process improves the quality of the results without additional costs.

In practice, the T_1 value also have an influence on the quality of the inferred paths. In addition, Figure 9 shows the paths obtained for an actual one-day trip, considering various T_1 values (note that the actual location names were replaced by letters). The best results are obtained with $T_1 = 15$ minutes, lesser values leading to unwanted pause detections between locations A and D. On the contrary, with a greater T_1, some pauses are not detected, e.g., the first pause at location B.

Network Message Load: Our second experiment assesses the actual gain achieved

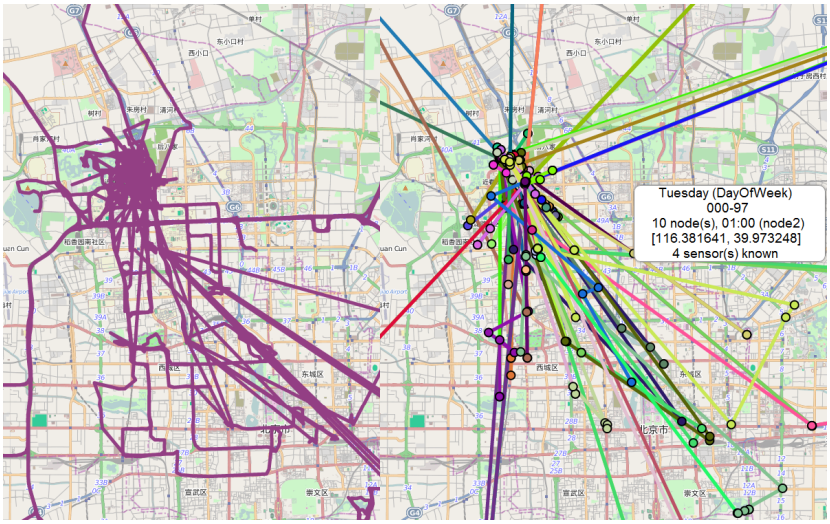

Figure 10: Mobility of the user 001 (left) and the extracted paths (right).

by the path prediction technique. As we mentioned in Section 3, we expect path prediction to reduce the number of updates exchanged with the registry when new sensors are discovered or lost while the user is moving. However, given that human mobility is not perfectly predictable [35] and repeatable, some corrections must be sent to the registry in any case. Sometimes, when the predicted path is completely wrong, the traffic induced by the corrections is higher than just naively sending the list of sensors discovered at each location. For example, this happens when users are at a location L_i while the registry expects them to be at the location L_j, with the distance between L_i and L_j being greater than the range of the sensors. In this case, the proxy must unregister all the wrong sensors and register all the sensors actually discovered. This is the role of the β threshold to alleviate the negative effects of this problem, by preventing SPINEL to send an unreliable prediction to the registry.

In order to evaluate the actual gain of our predictive approach in an environment with a significant number of sensors, we set up a simulation using: (i) a mobility dataset, and (ii) a set of generated virtual sensors in a defined area.

We used the mobility dataset opened by the GeoLife project [36]. The dataset involves 182 users equipped with a GPS device, and covers a period of five years. Most of the 17621 trajectories take place in Beijing, China. First, we cleaned the dataset by removing the users with a dataset covering a too short period. Users with huge discontinuities in their trajectories (more than one week) were removed as well. We then extracted the paths for the 64 remaining users, by detecting the pause (minimum 15 minutes) in their trajectories on a per-day basis (see Figure 10 for an example). For each user, half of the paths was used to train the path prediction algorithm and the other half was used for its evaluation.

Regarding the sensors, they were placed around points of interests (e.g., restaurants, supermarket, parking lots) provided by the collaborative mapping project Open Street Map (`openstreetmap.org`). In addition, the sensors were generated from realistic profiles, with various connectivity (Bluetooth, Wi-Fi, ZigBee) and communication ranges to simulate the effect of the environment (e.g., buildings) on wireless communication.

Using this environment, we compare the gain between the predictive and the

Require: P the path composed of the n last pauses of the user,
$\mathbb{P}$ the set of paths stored in the mobility database,
Q_{-1} the current prediction,
S_{-1} the set of sensors discovered at the previous location

$Q \leftarrow \varnothing, best \leftarrow 0$
for all $(x, p_i) \in \mathbb{P} \times P$ **do**
  $tmp \leftarrow \widetilde{\widetilde{S}}(x, P)$
  if $tmp \geq \beta$ **and** $tmp > best$ **then**
    $best \leftarrow tmp, Q \leftarrow x$
  end if
end for
if $Q = \varnothing$ **then**
  return
else if $Q \neq Q_{-1}$ **and** $\widetilde{\widetilde{S}}(Q, Q_{-1}) < \gamma$ **then**
  send a full correction message
else
  discover the set S of closeby sensors
  if $S \neq S_{-1}$ **then**
    register $S - (S \cap S_{-1})$, unregister $S_{-1} - (S \cap S_{-1})$
  end if
end if

Algo. 2: Predictive registration

Require: L the current location,
L_{-1} the previous location,
S_{-1} the set of sensors discovered at the location L_{-1}

discover the set S of closeby sensors

if $S \neq S_{-1}$ **then**
  register $S - (S \cap S_{-1})$
  unregister $S_{-1} - (S \cap S_{-1})$
end if

Algo. 3: Naïve registration

naive approach for registering sensors to the discovery system, described in Algorithms 2 and 3 respectively. The naïve approach simply registers the full set of sensors discovered at each location, while the predictive approach predicts the path that will be followed by the user and registers the sensors that are known to be available along this path. During the experiment, we measure the number $\mathcal{P}$ of messages sent to correct the prediction and the number $\mathcal{N}$ of messages sent by the naive approach. The gain is then computed as $\frac{\mathcal{N}-\mathcal{P}}{\mathcal{N}}$. Note that the gain becomes negative (overhead) when the prediction is significantly wrong, i.e., when the distance between the predicted and the actual locations is greater than the sensor range.

In order to evaluate the quality of our path prediction technique, the predictive registration described in Algorithm 2 is implemented using two path prediction techniques: our *Complete Path Prediction* (C2P) algorithm and an existing algorithm introduced by [9] for trajectory prediction in the context of data delivery to mobile sinks in WSNs. In a nutshell, the latter method clusterizes similar trajectories and aligns them using a modified DNA sequence alignment algorithm. For each cluster, a compact probabilistic representation is extracted and used to find the cluster to which the last moves of the user belong. Once a cluster is matched, sequence alignment is used to find the trajectories that partially match the last moves of the user and then predict the future locations. For our experiment purpose, we implemented this solution and adapted it for supporting trajectories composed of complex pauses (i.e., location, arrival time, pause time, azimuth and set of sensors) instead of simple locations.

Our experiment is repeated for each user in the dataset and the results are shown in Figure 11, for $\beta = 0$ (no threshold), $\beta = 0.7$ and the trajectory prediction technique of [9]. As we can see, even without a β threshold, the gain is greater than zero in average (~10%). As expected, prediction errors introduce overhead (for instance, up to 77% for user 026), but this overhead is reduced when the β threshold is set. As we can see, with a β of 0.7, there is a 23% gain compared to the naive approach. Regarding the existing trajectory prediction technique, we can see that the results are not as good as those of C2P. This is mainly due to the alignment phase, which tends

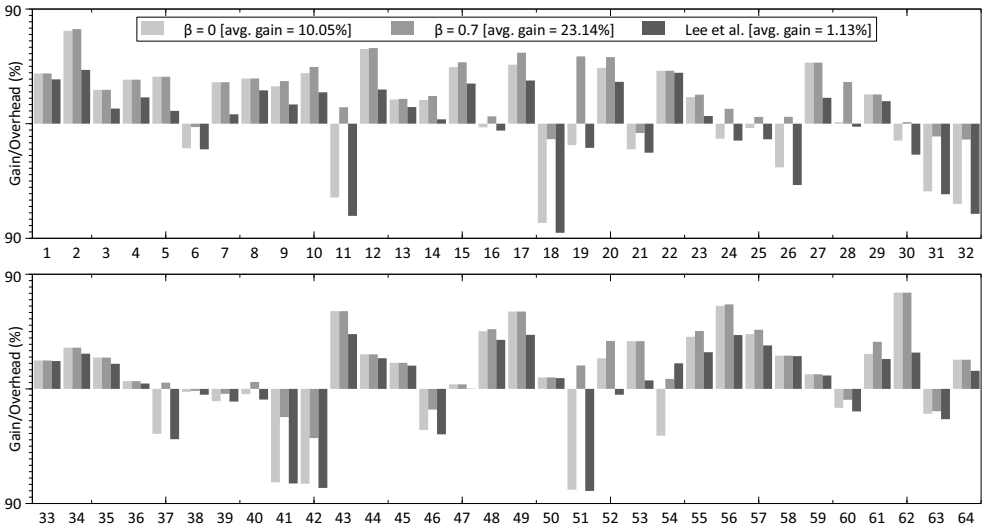

Figure 11: Average gain by user (percentage of messages saved).

to introduce noise in the trajectories and reduce the accuracy of the overall prediction. These small errors have a very strong impact on the number of corrections sent to the registry, due to the limited range of the sensors. Our simulations indeed showed that increasing the sensor communication range leads to better results.

5 Conclusion

We have introduced the concept of *opportunistic proxy*: mobile proxies carried by users' smartphones, which are able to discover closeby sensors while moving and present them seamlessly to the other IoT users and devices. We then presented Spinel, an opportunistic proxy that opens existing sensor networks to the IoT and, specifically, reduces the communication costs related to sensors' registration, using a novel path prediction techniques (C2P) to avoid useless message exchanges. In addition of opening legacy sensors to the IoT infrastructure, the users can benefit of the measurements acquired by Spinel from these sensors in order to have more accurate and fresh information regarding their environment.

We evaluated Spinel by implementing a prototype and assessing its performance. We showed that our mobility analysis process produces more reliable results by using the GPS when the mobility detection results are subject to uncertainty, without consuming more energy than an accelerometer-only approach. Regarding the communication load, we showed how our approach can reduce the number of messages exchanged for registration, according to the communication ranges of the sensors considered, thus reducing the network resources consumption.

From a technical perspective, we plan to extend this work in several directions in the future. First, we want to investigate the relevance of using a human mobility model [37, 38] to improve the results of the prediction process when past mobility data are not available (cold start), or when the past mobility does not give useful information (e.g., when the user moves a lot with very few repeatable patterns). Second, we want to extend our opportunistic proxy to open mobile sensors networks to the IoT, as our current approach is dedicated to static sensors networks already deployed (e.g., in smart cities). To this end, a mobility model can be used to infer

the probability that the user stays close enough to the mobile sensors, for a given period of time. Third, we want to investigate how to infer automatically the values of the thresholds and constants used in our approach. Mobility analysis and prediction techniques often introduce several customization parameters that must be fixed according to the use cases, the requirements and the goals considered. However, in our case, it would be possible to leverage learning and optimization techniques to find optimal or good-enough values for a specific usage, especially if the user is allowed to provide some feedback regarding the paths built during the mobility analysis. Finally, we want to integrate indoor positioning techniques in order to apply our prediction algorithm to areas where the GPS is not available. As the computation power and the energy resources of smartphones are limited, it is necessary to find a tradeoff between indoor positioning accuracy and resource consumption. Similarly, we would like to investigate how state-of-the-art learning techniques could improve the mobility detection process, especially regarding the noisy data produced by smartphones' accelerometers.

From an evaluation perspective, our energy consumption results are based on a single phone and we thus want to conduct experiments at a larger-scale to analyze quantitatively how much energy Spinel can save in practice, for various devices, operating systems, configurations and usages.

The last point we want to study is how Spinel can benefit the IoT, even if the 6LoWPAN/CoAP stack becomes dominant in the future, by being a mobility-aware edge layer between resource-constrained sensors and the IoT infrastructure. Use cases include, but are not limited to, predicting meeting points or destinations, harmonizing discovery mechanisms if several competing protocols emerge, and providing anonymization or caching capabilities.

References

[1] Luca Mottola and Gian Pietro Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Computing Survey*, 43(3):19–76, 2011.

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[3] Zhiwei Yan, Ning Kong, Ye Tian, and Yong-Jin Park. A universal object name resolution scheme for IoT. In *Proc. of the 3rd International Conference on Green Computing and Communications/Internet of Things/Cyber, Physical and Social Computing*, pages 1120–1124, 2013.

[4] Sara Hachem, Animesh Pathak, and Valerie Issarny. Service-oriented middleware for large-scale mobile participatory sensing. *Pervasive and Mobile Computing*, 10(1):66–82, 2014.

[5] Geoff Mulligan. The 6LoWPAN architecture. In *Proc. of the 4th workshop on Embedded networked sensors*, pages 78–82, 2007.

[6] Zach Shelby, Klaus Hartke, and Carsten Bormann. RFC 7252 - constrained application protocol (CoAP). `tools.ietf.org/html/rfc7252`, 2014.

[7] Sara Hachem, Vivien Mallet, Ventura Raphaël, Pierre-Guillaume Raverdy, Animesh Pathak, Valérie Issarny, and Rajiv Bhatia. Monitoring noise pollution using

the urban civics middleware. In *Proc. of the 1st International Conference on Big Data Computing Service and Applications*, BDS '15, pages 52–61, 2015.

[8] Marta C Gonzalez, Cesar A Hidalgo, and Albert-Laszlo Barabasi. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, 2008.

[9] HyungJune Lee, Martin Wicke, Branislav Kusy, Omprakash Gnawali, and Leonidas Guibas. Data stashing: Energy-efficient information delivery to mobile sinks through trajectory prediction. In *Proc. of the 9th International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 291–302, 2010.

[10] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008.

[11] Adam Dunkels, Juan Alonso, Thiemo Voigt, Hartmut Ritter, and Jochen Schiller. Connecting wireless sensornets with TCP/IP networks. In *Wired/Wireless Internet Communications*, pages 143–152. Springer, Berlin, 2004.

[12] Liam McNamara, Cecilia Mascolo, and Licia Capra. Media sharing based on colocation prediction in urban transport. In *Proc. of the 14th International Conference on Mobile Computing and Networking*, pages 58–69, 2008.

[13] Vladimir Vukadinović, Ólafur Ragnar Helgason, and Gunnar Karlsson. An analytical model for pedestrian content distribution in a grid of streets. *Mathematical and Computer Modelling*, 57(11-12):2933–2944, 2013.

[14] Guangtao Xue, Yuan Luo, Jiadi Yu, and Minglu Li. A novel vehicular location prediction based on mobility patterns for routing in urban VANET. *Journal on Wireless Communications and Networking*, 2012(1):222–236, 2012.

[15] Taebok Yoon and Jee-Hyong Lee. Goal and path prediction based on user's moving path data. In *Proc. of the 2nd Conference on Ubiquitous Information Management and Communication*, pages 475–480, 2008.

[16] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava. Participatory sensing. In *Proc. of the 4th Workshop on World-Sensor-Web: Mobile Device Centric Sensor Networks and Applications*, pages 225–230, 2006.

[17] Nicholas D. Lane, Shane B. Eisenman, Mirco Musolesi, Emiliano Miluzzo, and Andrew T. Campbell. Urban sensing systems: Opportunistic or participatory? In *Proc. of the 9th Workshop on Mobile Computing Systems and Applications*, pages 11–16, 2008.

[18] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Cloudlets: Bringing the cloud to the mobile user. In *Proc. of the 3rd ACM Workshop on Mobile Cloud Computing and Services*, pages 29–36, 2012.

[19] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: Making smartphones last longer with code offload. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 49–62, 2010.

[20] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and Kiryong Ha. The role of cloudlets in hostile environments. *Pervasive Computing, IEEE*, 12(4):40–49, 2013.

[21] Soumya Simanta, Grace A. Lewis, Ed Morris, Kiryong Ha, and Mahadev Satyanarayanan. A reference architecture for mobile code offload in hostile environments. In *Proc. of the 9th Working IEEE/IFIP Conference on Software Architecture*, pages 282–286, 2012.

[22] Mohsen Sharifi, Somayeh Kafaie, and Omid Kashefi. A survey and taxonomy of cyber foraging of mobile devices. *Communications Surveys Tutorials, IEEE*, 14(4):1232–1243, 2012.

[23] EPCglobal. The GS1 EPCglobal architecture framework v1.6. `www.gs1.org/gsmp/kc/epcglobal/architecture`, 2014.

[24] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an acquisitional query processing system for sensor networks. *Transactions on Database Systems*, 30(1):122–173, 2005.

[25] Benjamin Billet and Valérie Issarny. Dioptase: A distributed data streaming middleware for the future web of things. *Journal of Internet Services and Applications*, 5(13):13–32, 2014.

[26] Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. Energy-efficient rate-adaptive GPS-based positioning for smartphones. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, pages 299–314, 2010.

[27] A.M. Khan, Y. K Lee, S.Y. Lee, and T. S Kim. Human activity recognition via an accelerometer-enabled-smartphone using kernel discriminant analysis. In *Proc. of the 5th International Conference on Future Information Technology*, pages 1–6, 2010.

[28] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *Proc. of the 11th ACM Conference on Embedded Networked Sensor Systems*, pages 13:1–13:14, 2013.

[29] Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kjærgaard, Anind Dey, Tobias Sonne, and Mads Møller Jensen. Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition. In *Proc. of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 127–140, 2015.

[30] Paul Baumann, Wilhelm Kleiminger, and Silvia Santini. How long are you staying?: Predicting residence time from human mobility traces. In *Proc. of the 19th International Conference on Mobile Computing & Networking*, MobiCom '13, pages 231–234, 2013.

[31] M. Papandrea, M. Zignani, S. Gaito, S. Giordano, and G.P. Rossi. How many places do you visit a day? In *Proc. of the 11th International Conference on Pervasive Computing and Communications Workshops*, pages 218–223, 2013.

[32] Charles F. F. Karney. Algorithms for geodesics. *Journal of Geodesy*, 87(1):43–55, 2013.

[33] Alberto Coen-Porisini, Pietro Colombo, and Sabrina Sicari. Dealing with anonymity in wireless sensor networks. In *Proc. of the 25th ACM Symposium on Applied Computing*, pages 2216–2223, 2010.

[34] Niels Brouwers and Koen Langendoen. Pogo, a middleware for mobile phone sensing. In *Proc. of the 13th International Middleware Conference*, pages 21–40, 2012.

[35] Xin Lu, Erik Wetter, Nita Bharti, Andrew J. Tatem, and Linus Bengtsson. Approaching the limit of predictability in human mobility. *Scientific Reports*, 3(2923):44–53, 2013.

[36] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from GPS trajectories. In *Proc. of the 18th International Conference on World Wide Web*, pages 791–800, 2009.

[37] Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, Seong Joon Kim, and Song Chong. On the levy-walk nature of human mobility. *Transactions on Networking*, 19(3):630–643, 2011.

[38] G.S. Thakur and A. Helmy. COBRA: A framework for the analysis of realistic mobility models. In *Proc. of the 32th International Conference on Computer Communications*, pages 3351–3356, 2013.